STIC EIC 2100 Search Request Form

130464

86

Today's Date: 8/23/04

What date would you like to use to limit the search?

Priority Date: 12/19/2000 Other:

Name Trenton Roche

AU 2124 Examiner # 79908

Room # PK2-5D40 Phone 305-4627

Serial # 09/740,601

Format for Search Results (Circle One):

(PAPER) DISK EMAIL

Where have you searched so far?

(USP) (DWPI) (EPO) (JPO) (ACM) (IBM TDB)

(IEEE) INSPEC SPI Other

Is this a "Fast & Focused" Search Request? (Circle One) (YES) NO

A "Fast & Focused" Search is completed in 2-3 hours (maximum). The search must be on a very specific topic and meet certain criteria. The criteria are posted in EIC2100 and on the EIC2100 NPL Web Page at http://ptoweb/patents/stic/stic-tc2100.htm.

What is the topic, novelty, motivation, utility, or other specific details defining the desired focus of this search? Please include the concepts, synonyms, keywords, acronyms, definitions, strategies, and anything else that helps to describe the topic. Please attach a copy of the abstract, background, brief summary, pertinent claims and any citations of relevant art you have found.

Debugger in a multi-processor (multi-cell) computer system, wherein there are provided **multiple symbol tables** divided among the plurality of processors. This system provides a way to select a corresponding symbol table from among the multiple symbol tables by checking to see if an address in an address pointer falls within the specified range of addresses in a symbol table. Involves first searching a base (default) symbol table, then moving on to check "offset" symbol tables.

Important aspects:
Debugger
Multiple symbol tables
Selecting one of the symbol tables

STIC Searcher David Holloway Phone 308-7794

Date picked up 8-23-04 Date Completed 8-23-04

```
Set     Items   Description
S1       8182   (SYMBOL? ? OR SYMBOLIC()NAME? OR VARIABLE?)(2N)(TABLE? OR -
            GRID OR MATRIX OR MATRICE? OR TUPLE? OR ROW(N)COLUMN?)
S2    3502715   DEBUG? OR EMULATOR? OR SIMULAT? OR MODELLING
S3        139   S1(2N)(SELECT? OR CHOOSE? OR IDENTIF? OR FIND?)
S4        129   S1(2N)(MULTIPL? OR PLURAL? OR SEVERAL? OR VARIOUS OR VARIE-
            T? OR MANY)
S5       2116   S2(2N)(MULTIPROCESSOR? OR (MULTIPL? OR PLURAL? OR SEVERAL?
            OR VARIOUS OR VARIET?)(N)PROCESSOR?)
S6         39   S2 AND (S3 OR S4)
S7          2   S5 AND S1
S8         41   S6 OR S7
S9         34   RD (unique items)
S10        24   S9 NOT PY>2000
File   8:Ei Compendex(R) 1970-2004/Aug W3
         (c) 2004 Elsevier Eng.  Info. Inc.
File  35:Dissertation Abs Online 1861-2004/Jul
         (c) 2004 ProQuest Info&Learning
File 202:Info. Sci. & Tech. Abs. 1966-2004/Jul 12
         (c) 2004 EBSCO Publishing
File  65:Inside Conferences 1993-2004/Aug W4
         (c) 2004 BLDSC all rts. reserv.
File   2:INSPEC 1969-2004/Aug W3
         (c) 2004 Institution of Electrical Engineers
File  94:JICST-EPlus 1985-2004/Aug W1
         (c)2004 Japan Science and Tech Corp(JST)
File 111:TGG Natl.Newspaper Index(SM) 1979-2004/Aug 23
         (c) 2004 The Gale Group
File 233:Internet & Personal Comp. Abs. 1981-2003/Sep
         (c) 2003 EBSCO Pub.
File   6:NTIS 1964-2004/Aug W3
         (c) 2004 NTIS, Intl Cpyrght All Rights Res
File 144:Pascal 1973-2004/Aug W3
         (c) 2004 INIST/CNRS
File 434:SciSearch(R) Cited Ref Sci 1974-1989/Dec
         (c) 1998 Inst for Sci Info
File  34:SciSearch(R) Cited Ref Sci 1990-2004/Aug W3
         (c) 2004 Inst for Sci Info
File  62:SPIN(R) 1975-2004/Jun W4
         (c) 2004 American Institute of Physics
File  99:Wilson Appl. Sci & Tech Abs 1983-2004/Jul
         (c) 2004 The HW Wilson Co.
File  95:TEME-Technology & Management 1989-2004/Jun W1
         (c) 2004 FIZ TECHNIK
```

10/5/15 **(Item 5 from file: 2)**
DIALOG(R)File 2:INSPEC

02354980 INSPEC Abstract Number: C85002831
Title: FORTRAN is FORTRAN is FORTRAN
Author(s): Kaufman, L.; Massart, D.L.
Journal: TRAC Trends in Analytical Chemistry vol.3, no.7 p.172
Publication Date: Aug. 1984 Country of Publication: Netherlands
CODEN: TTAEDJ ISSN: 0165-9936
U.S. Copyright Clearance Center Code: 0165-9936/84/$02.00
Language: English Document Type: Journal Paper (JP)
Treatment: Applications (A); General, Review (G)
Abstract: One of the major problems for using computer programs in a research environment is that of transferring a program written for one computer system to another computer or to the same computer using a different operating system or different compiler. A very elegant solution exists for FORTRAN which is the most widely used scientific language. The PFORT Verifier is a program which checks whether the statements of a FORTRAN program (whether it is FORTRAN 77, IV, extended, etc.) belong to PFORT, a subset of American National Standard FORTRAN (ANS FORTRAN). PFORT also diagnoses several errors which compilers often miss (relating to interprogram-unit communication and to COMMON statements). It also provides a number of features useful in **debugging** and documentation of a program. It produces error diagnostics, **symbol tables** and **several** lists of cross-references. An interprogram-unit description includes for each unit a list of variables, common blocks, program units called, and the units which call them. A list of common definitions is also produced. The PFORT Verifier is itself written in PFORT and can easily be installed on a variety of computers using two small assembly language problems. (3 Refs)
Subfile: C
Descriptors: FORTRAN
Identifiers: Fortran IV; error diagnosis; FORTRAN; computer programs; operating system; compiler; scientific language; PFORT Verifier; FORTRAN 77 ; ANS FORTRAN; interprogram-unit communication; COMMON statements; **debugging** ; documentation; error diagnostics; symbol tables; variables; common blocks; assembly language problems
Class Codes: C6140D (High level languages)

```
Set     Items   Description
S1        116   (MULTIPLE OR PLURAL OR SEVERAL OR MULTIPLICITY OR PLURALIT-
            Y)(3N)SYMBOL()TABLE?
S2         77   S1 NOT PY>2000
S3         77   S2 NOT AD>20001219
S4         20   S3 AND (DEBUG? OR EMULAT? OR DE()(BUG OR BUGGER))
S5        261   (SELECT? OR CHOOSE? OR FIND? OR LOCAT? OR IDENTIF?)(2N)SYM-
            BOL()TABLE?
S6         32   S3 AND S5
S7         49   S4 OR S6
S8         49   S7 NOT PD=20001219:20021219
S9         49   S8 NOT PD=20021219:20040901
File 621:Gale Group New Prod.Annou.(R) 1985-2004/Aug 23
         (c) 2004 The Gale Group
File 545:Investext(R) 1982-2004/Aug 23
         (c) 2004 Thomson Financial Networks
File 654:US Pat.Full. 1976-2004/Aug 19
         (c) Format only 2004 The Dialog Corp.
File 440:Current Contents Search(R) 1990-2004/Aug 23
         (c) 2004 Inst for Sci Info
File 351:Derwent WPI 1963-2004/UD,UM &UP=200453
         (c) 2004  Thomson Derwent
File 349:PCT FULLTEXT 1979-2002/UB=20040819,UT=20040812
         (c) 2004 WIPO/Univentio
File 348:EUROPEAN PATENTS 1978-2004/Aug W03
         (c) 2004 European Patent Office
File 347:JAPIO Nov 1976-2004/Apr(Updated 040802)
         (c) 2004 JPO & JAPIO
File 275:Gale Group Computer DB(TM) 1983-2004/Aug 23
         (c) 2004 The Gale Group
File 141:Readers Guide 1983-2004/Jul
         (c) 2004 The HW Wilson Co
File  95:TEME-Technology & Management 1989-2004/Jun W1
         (c) 2004 FIZ TECHNIK
File  88:Gale Group Business A.R.T.S. 1976-2004/Aug 20
         (c) 2004 The Gale Group
File  47:Gale Group Magazine DB(TM) 1959-2004/Aug 23
         (c) 2004 The Gale group
File  34:SciSearch(R) Cited Ref Sci 1990-2004/Aug W3
         (c) 2004 Inst for Sci Info
File  16:Gale Group PROMT(R) 1990-2004/Aug 23
         (c) 2004 The Gale Group
File  15:ABI/Inform(R) 1971-2004/Aug 23
         (c) 2004 ProQuest Info&Learning
File   2:INSPEC 1969-2004/Aug W3
         (c) 2004 Institution of Electrical Engineers
```

9/5/25 (Item 24 from file: 654)
DIALOG(R)File 654:US Pat.Full.

3510244 **IMAGE Available
Derwent Accession: 1991-009335
Utility
REASSIGNED
E/ Compiler using clean lines table with entries indicating unchanged text lines for incrementally compiling only changed source text lines
Inventor: McKeeman, William M., Hollis, NH
Aki, Shota, Weare, NH
Assignee: Digital Equipment Corporation (02), Maynard, MA
Digital Equipment Corp (Code: 23989)
Examiner: Lee, Thomas C. (Art Unit: 237)
Combined Principal Attorneys: Young, Barry N.; Cefalo, Albert P.

	Publication Number	Kind	Date	Application Number	Filing Date
Main Patent	US 5325531	A	19940628	US 92819611	19920109
Continuation	Abandoned			US 89375401	19890630

Disclaimer Date: 20100126
Current US Classification (Main): 717008000 (X-ref): 717010000
US Classification on document (Main): 395700 (X-ref): 3642804; 3642592; 364DIG1
International Classification (Edition 1): G06F-009/45
Examiner Field of Search (US): 364DIG1; 364DIG2; 364200; 364900; 395700

Cited US Patents:

Patent Number	Date YYYYMM	Main US Class	Inventor
US 3711863	197301	371019	Bloom
US 4204253	198005	395575	van den Hanenberg
US 4398249	198308	395700	Pardo
US 4463423	198407	395700	Potash
US 4558413	198512	395700	Schmidt
US 4589068	198605	395700	Heinen, Jr.
US 4667290	198705	395700	Goss
US 4686623	198708	395700	Wallace
US 4720778	198801	395575	Hall
US 4734854	198803	395700	Afshar
US 4809170	198902	395700	Leblang
US 4819233	198904	371019	Delucia
US 4833606	198905	395700	Iwasawa
US 4931928	199006	395600	Greenfeld
US 4951192	199008	395700	Chase, Jr.
US 5065400	199111	395700	Masuishi
US 5133063	199207	395500	Naito
US 5170465	199212	395700	McKeeman
US 5182806	199301	395700	McKeeman
US 5193191	199303	395700	McKeeman
US 5201050	199304	395700	McKeeman

Cited non-US Patents:

Patent Number	Date YYYYMM	Main US Class	Main IPC
FR 2533721	198403		

Cited non-Patent References:

Aho, A. V., et al., Compilers: Principles, Techniques, and Tools, Addison-Wesley Publishing Co., pp. 1-15 (1986).
Fritzon, P., "Preliminary Experience from the DICE System--a Distributed Incremental Compiling Environment," ACM SIGPLAN Notices, vol. 19, No. 5, pp. 113-123 (May 1984).

Medina-Mora, R., et al., "An Incremental Programming Environment," IEEE Transactions on Software Engineering, vol. SE-7, No. 5, pp. 472-482 (Sep. 1981).
Ryan, J. L., et al., "A Conversational System for Incremental Compilation and Execution in a Time-Sharing Environment," AFIPS Conference Proceedings, vol. 29, Fall Joint Computer Conference, pp. 1-21 (1966).
Reiss, S. P., "An Approach to Incremental Compilation," ACM SIGPLAN Notices, vol. 19, No. 6, pp. 144-151 (Jun. 1984).
Schwartz, M. D., et al., "Incremental Compilation in Magpie," ACM SIGPLAN Notices, vol. 19, No. 6, pp. 122-131 (Jun. 1984).
Fritzon, P., "Symbolic Debugging Through Incremental Compilation in an Integrated Environment," The Journal of Systems and Software 3, pp. 285-294 (1983).
Fritzon, P., "Fine-Grained Incremental Compilation for Pascal-like Languages," Software Systems Research Center, Linkoping Institute of Technology, S-581 83 Linkoping, Sweden, LiTH-MAT-R-82-15, pp. 1-32 (Jul. 1982).
Ayres, R. B., et al., "Partial Recompilation," AFIPS Conference Proceedings, vol. 38, Spring Joint Computer Conference, pp. 497-502 (1971).
Earley, J. et al., "A Method for Incrementally Compiling Languages with Nested Statement Structure," Communications of the ACM, vol. 15, No. 12, pp. 1040-1044 (Dec. 1972).
Fritzon, P., "A Systematic Approach to Advanced Debugging through Incremental Compilation," ACM SIGPLAN Notices, vol. 18, No. 8, pp. 130-139 (Aug. 1983).
Fritzon, P., Towards a Distributed Programming Environment Based on Incremental Compilation, PhD Thesis, Dept. of Computer and Information Science, Linkoping Univ., Linkoping, Sweden, pp. 59-100 (1984).
Feldman, S. I., "Make--A Program For Maintaining Computer Programs," Software--Practice and Experience, vol. 9, pp. 255-265 (1979).
Waite, William M. and Gerhard Goos, Compiler Construction, Springer-Verlag, New York, 1984, Section 1.4, pp. 8-9.
Walker et al., "The Symbolics Genera Programming Environment", IEEE Software, vol. 4, No. 6, Nov. 1987, pp. 36-44.
Sebesta, "Conversational programming systems", Journal of Pascal, Ada & Modula-2, vol. 4, No. 3, May/Jun. 1985, pp. 9-22.
Adams, "SUNPRO: engineering, a practical program development environment," Proc. of An Int'l Workshop Adv. Prog. Environments, Jun. 16, 1986, pp. 86-96.
Alberga et al., "A program development tool", IBM Journal of Research and Development, vol. 28, No. 1, Jan. 1984, pp. 60-72.
Reiss, "PECAN: program development systems that support multiple views," IEEE Trans. on Software Engineering, vol. SE-11, No. 3, Mar. 1985, pp. 276-285.

Fulltext Word Count: 16183
Number of Claims: 9
Exemplary or Independent Claim Number(s): 1
Number of Drawing Sheets: 16
Number of Figures: 17
Number of US cited patent references: 21
Number of non-US cited patent references: 1
Number of non-patent cited references: 19

Post Issue Legal Status:
Calculated Expiration Date: 20110628

Reassignment:
Recorded: 20020109
Action: ASSIGNMENT OF ASSIGNORS INTEREST
Assignor: DIGITAL EQUIPMENT CORPORATION DATE SIGNED: 12/09/1999
COMPAQ COMPUTER CORPORATION DATE SIGNED: 06/20/2001
Assignee: COMPAQ INFORMATION TECHNOLOGIES GROUP, L.P. 20555 STATE HIGHWAY 249 HOUSTON TEXAS 77070
Reel: 012447
Frame: 0903
Contact: CONLEY, ROSE & TAYON, P.C. JONATHAN M. HARRIS P.O. BOX 3267

HOUSTON, TEXAS 77253-3267

Recorded: 20031103
Action: CHANGE OF NAME
Assignor: COMPAQ INFORMANTION TECHNOLOGIES GROUP LP DATE SIGNED: 10/01/2002
Assignee: HEWLETT-PACKARD DEVELOPMENT COMPANY, L.P. 20555 SH 249 HOUSTON TEXAS 77070
Reel: 014102
Frame: 0224
Contact: HEWLETT-PACKARD COMPANY RECORDS MANGER INTELLECTUAL PROPERTY ADMINSITRATION P.O. BOX 272400 FORT COLLINS, CO 80527-2400

Abstract:
A computer-aided software development system includes programs to implement edit, compile, link and run sequences, all from memory, at very high speed. The compiler operates on an incremental basis, line-by-line, so if only one line is changed in an edit session, then only that line need be recompiled if no other code is affected. Dependency analysis is performed incrementally, without requiring the user to enter dependencies. Scanning is also done incrementally, and the resulting token list saved in memory to be used again where no changes are made. All of the linking tables are saved in memory so there is no need to generate link tables for increments of code where no changes in links are needed. The parser is able to skip lines or blocks of lines of source code which haven't been changed. All of the source code text modules, the token lists, symbol tables, code tables and related data saved from one compile to another are maintained in virtual memory rather than in files so that speed of operation is enhanced. Also, the object code created is maintained in memory rather than in a file, and executed from this memory image, to reduce delays. A virtual memory management arrangement for the system assures that all of the needed data modules and code is present in real memory by page swapping, but with a minimum of page faults, again to enhance operating speed.

What is claimed is:
Exemplary or Independent Claim(s):
1. A method of operating a computer for recompiling source code to produce object code in a plurality of object code tables, said source code having been previously compiled to produce a plurality of object code tables, comprising the steps of:
a) editing by said computer a plurality of modules of said source code using an editor, each of said modules containing a plurality of lines of said source code, said source code being stored by said computer in memory in a plurality of source-text buffers; and storing by said computer for each buffer an identification of selected lines of said plurality lines of said source code, said selected lines being lines that have been changed by said editing for each buffer; and, during said step of editing, generating by said computer a clean-lines table having an entry for each of said lines of each of said source-text buffers, said entry including an indication of how many following lines have not been changed before reaching the next following line that has been changed;
b) performing by said computer an incremental dependency analysis on said modules of source code in said plurality of source-text buffers while said buffers are in said memory, said dependency analysis including:
finding in each of said buffers lines having statements depending upon statements in said selected lines which have been changed, as determined by said stored identification for each buffer, to thereby select in said plurality of buffers only lines which are dependent on said selected lines which have been changed and therefore must be recompiled;
c) incrementally recompiling by said computer said selected lines which have been changed in said plurality of source-text buffers, as determined by said identification for each of said buffers, and said lines in said plurality of source-text buffers having statements depending upon statements in said selected lines which have been

changed, to update by said computer said plurality of object code tables, where each buffer has a corresponding one of said plurality of object code tables; said step of recompiling including checking said entry for each line to determine how many lines can be skipped in recompiling.

Non-exemplary or Dependent Claim(s):

2. A method according to claim 1 including the steps of, after said step of recompiling:
saving by said computer the contents of said source-text buffers and said code tables in said memory,
and, thereafter, again performing said steps (a), (b), and (c) by said computer, re-using said saved source-text buffers and said saved code tables.

3. A method according to claim 1 wherein said identification includes for each line of each one of said plurality of said source-text buffers an associated change-tag bit to indicate whether or not each line of said source code was changed during said editing.

4. A method according to claim 1 wherein said recompiling includes generating code for said object code tables for executable statements in said source code and includes creating symbol tables in said memory for declarative statements in said source code, each of said **symbol tables** containing a **plurality** of entries with each entry corresponding to one of said declarative statements, said symbol tables and said object code tables together comprising said produced object code.

5. A method of operating a computer for editing and recompiling source code, said source code having been previously compiled to produce object code, comprising the steps of:
a) editing by said computer a plurality of modules of said source code to change selected lines of said source code in said modules, and after said editing storing each module of said source code by said computer in memory in a separate one of a plurality of source-text buffers, each of said modules containing a plurality of lines of said source code and some of said lines being changed by said editing while some of said lines not being changed by said editing, each line in each module having an associated change-tag bit and said step of editing setting said change-tag bits to indicate whether or not each line was changed during said editing of each module; and, during said step of editing, generating by said computer a clean-lines table having an entry for each of said lines of each of said source-text buffers, said entry including an indication of how many following lines have not been changed before reaching the next following line that has been changed;
b) performing an incremental dependency analysis on said source code by said computer in said plurality of source-text buffers while said buffers are in said memory, said dependency analysis including:
finding in each of said buffers lines having statements depending upon statements in said lines which have been changed in said step of editing, as determined by said change-tag bits, to thereby select in said buffers only lines which are dependent on said lines which have been changed and therefore must be recompiled;
c) incrementally recompiling by said computer said lines which have been changed as indicated by said change-tag bits and recompiling said selected lines having depending statements, to generate updated code to replace parts of said object code in memory for each buffer; said step of recompiling including checking said entry for each line to determine how many lines can be skipped in recompiling6. A method according to claim 5 including the steps of saving said source-text buffers and said object code by said computer in said memory after said step of recompiling, and again performing steps (a), (b), and (c) by said computer, re-using said saved source-text buffers and said saved object code.

7. A method according to claim 5 including the step of halting said step of recompiling to return an error indication if an error is detected.

8. A method of operating a computer for editing and incrementally recompiling source code, said source code having been previously compiled to produce object code, comprising the steps of:

a) editing by said computer a plurality of modules of said source code using an editor, each of said modules containing a plurality of lines of said source code and some of said lines being changed by said editing while other ones of said lines remain unchanged, each one of said modules being stored by said computer in memory in one of a plurality of separate source-text buffers; and generating an indication of lines in each one of said source-text buffers that have been changed by said step of editing;
b) performing by said computer an incremental dependency analysis on said source code in said plurality of source-text buffers while said buffers are in said memory, said dependency analysis including: finding in each of said buffers lines having statements depending upon statements in said lines which have been changed by said step of editing, as determined by said indication, to thereby select in said buffers lines which are dependent on said lines which have been changed and therefore must be recompiled;
c) incrementally recompiling by said computer said lines in said source-text buffers which have been changed as determined from said indication and the lines in said source-text buffers having depending statements, said step of recompiling generating object code to produce in said memory a plurality of updated object code tables, one for each source-text buffer; said updated object code tables including new object code for said lines which have been changed and said lines having depending statements and including reused object code for said lines which remain unchanged and do not have depending statements;
d) said step of recompiling being halted by said computer to return an error indicator if an error is detected;
e) said step of recompiling including, for each one of said changed lines and said lines having depending statements in said source-text buffers;
i) scanning said source code by said computer to generate in said memory a table of lexical increments corresponding to lines of said source code,
ii) generating said object code by said computer to build said code tables from executable statements in said source code corresponding to said lexical increments;
iii) generating a symbol table by said computer in said memory from declarative statements in said lexical increments; and f) prior to said step of recompiling, the step of generating by said computer in said memory for each source-text buffer a clean-lines table including for each line of said source code a table entry; said table entry having a locator for locating each line in each source-text buffer, and having an indication of the number of unchanged lines following each line before reaching a changed line; said step of recompiling including checking said clean-lines table for each source-text buffer to determine which lines can be skipped.

9. A method according to claim 8 wherein said indication, in each one of said source-text buffers, includes a change-tag bit for each line of said source code indicating whether or not each line was changed by said editing.

9/5/34 **(Item 1 from file: 351)**
DIALOG(R)File 351:Derwent WPI

007887475 **Image available**
WPI Acc No: 1989-152587/198921
XRPX Acc No: N89-116441
Code debugging **computer system using optimising compiler - specifies ranges within compiled code when user resources reside in specified locations**
Patent Assignee: HEWLETT-PACKARD CO (HEWP)
Inventor: COUTANT D S; MELOY S A
Number of Countries: 006 Number of Patents: 005
Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week	
EP 317080	A	19890524	EP 88309777	A	19881019	198921	B
US 4953084	A	19900828	US 87121311	A	19871116	199037	
CA 1293810	C	19911231				199208	
EP 317080	B1	19950614	EP 88309777	A	19881019	199528	
DE 3853981	G	19950720	DE 3853981	A	19881019	199534	
			EP 88309777	A	19881019		

Priority Applications (No Type Date): US 87121311 A 19871116
Cited Patents: 2.Jnl.Ref; A3...9103; EP 229245; No-SR.Pub
Patent Details:

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
EP 317080	A	E	26		
	Designated States (Regional): DE FR GB IT				
EP 317080	B1	E	29	G06F-011/00	
	Designated States (Regional): DE FR GB IT				
DE 3853981	G			G06F-011/00	Based on patent EP 317080

Abstract (Basic): EP 317080 A

The computing system has a **debug** symbol table (55) which includes descriptions of each user resource in source code (41). Additionally, a range table is constructed. The range table contains, for each user resource which is stored in numerous locations during execution of the code, a list of ranges and a description of where the user resource may be found during each range. If the user resource is stored as a constant during a particular range, the value of the constant may be stored in the range table.

The description of each user resource in the **debug** symbol table includes a flat which indicates whether there is a list of ranges in the range table for a particular user resource. If there is the description of the particular user will include a pointer to the list of ranges for that user resource.

1B/11

Title Terms: CODE; **DEBUG** ; COMPUTER; SYSTEM; OPTIMUM; COMPILE; SPECIFIED; RANGE; COMPILE; CODE; USER; RESOURCE; SPECIFIED; LOCATE
Derwent Class: T01
International Patent Class (Main): G06F-011/00
International Patent Class (Additional): G06F-009/44
File Segment: EPI

9/5/43 (Item 2 from file: 275)
DIALOG(R)File 275:Gale Group Computer DB(TM)

01723364 SUPPLIER NUMBER: 16314515 (USE FORMAT 7 OR 9 FOR FULL TEXT)
An event-based, retargetable debugger . **(HP Distributed** Debugging **Environment) (includes related articles on compiler optimization and** debugging **and an introduction to** debugger **internals)**
Iyengar, Arun K.; Grzesik, Thaddeus S.; Ho-Gibson, Valerie J.; Hoover, Tracy A.; Vasta, John R.
Hewlett-Packard Journal, v45, n6, p33(11)
Dec, 1994
ISSN: 0018-1153 LANGUAGE: ENGLISH RECORD TYPE: FULLTEXT; ABSTRACT
WORD COUNT: 8295 LINE COUNT: 00658

ABSTRACT: The HP Distributed **Debugging** Environment (DDE) is a software **debugger** that provides event-based **debugging** of conventional and optimized programs executing on remote hosts running a variety of operating system. The **debugger** can **debug** programs written in several assembly languages plus C, C++, Pascal and Fortran. HP DDE features an X Window System- and OSF/Motif-based customizable GUI with multiple windows, context-sensitive pop-up menus and online help. A modular architecture enables the isolation of managers of language-, object code-, target- and user interface-dependent functions for specific platforms from the main generic **debugging** functions. HP DDE's architecture, comparisons to other event-based and optimized code **debuggers** , and the implementation and porting of HP DDE are discussed.

SPECIAL FEATURES: illustration; chart; program
COMPANY NAMES: Hewlett-Packard Co.--Products
DESCRIPTORS: **Debugging** /Testing Software; Software Design
SIC CODES: 7372 Prepackaged software
TICKER SYMBOLS: HWP
TRADE NAMES: HP Distributed **Debugging** Environment (**Debugging** /testing software)--Design and construction
OPERATING PLATFORM: HP/UX; Solaris; OSF/1
FILE SEGMENT: CD File 275

9/5/45 (Item 4 from file: 275)
DIALOG(R)File 275:Gale Group Computer DB(TM)

01467305 SUPPLIER NUMBER: 11872737 (USE FORMAT 7 OR 9 FOR FULL TEXT)
Keeping up with the Kahns. (Borland International Inc.'s C++ version 3.0 optimizing C++ compiler) (Software Review) (Evaluation)
Kemp, Paul
EXE, v6, n7, p28(5)
Dec, 1991
DOCUMENT TYPE: Evaluation ISSN: 0268-6872 LANGUAGE: ENGLISH
RECORD TYPE: FULLTEXT; ABSTRACT
WORD COUNT: 2402 LINE COUNT: 00188

ABSTRACT: Borland International Inc's C++ version 3.0 optimizing C++ compiler includes Turbo C++ for Windows; the compiler complies with AT&T's C++ version 2.1 language specification but also supports version 3.0 templates. The command-line compiler, linker and text-mode integrated development environment (IDE) are hosted under DOS Protected Mode Interface (DPMI), which means that the DOS-extended versions of the utilities have been replaced by new versions that use all available memory on the host machine. Turbo **Debugger** now supports **debugging** of C++ 2.1 features, including nested classes; it returns correct values for variables enregistered by the optimizer and will not return values for variables eliminated by the optimizer. The many new features and enhancements in the program make it a significant upgrade; Microsoft's C++ products have a lot of catching up to do.

SPECIAL FEATURES: illustration; table
COMPANY NAMES: Borland International Inc.--Products
DESCRIPTORS: Compiler; Evaluation; C Programming Language; Object-Oriented Programming
SIC CODES: 7372 Prepackaged software
TICKER SYMBOLS: BORLN
TRADE NAMES: Borland C++ 3.0 (Compiler)--evaluation; Borland Turbo C++ for Windows (Compiler)--evaluation
OPERATING PLATFORM: MS Windows
FILE SEGMENT: CD File 275

9/5/46 (Item 1 from file: 141)
DIALOG(R)File 141:Readers Guide

00555166 H.W. WILSON RECORD NUMBER: BRGA85055166
TLC-Lisp.
Wong, William, 1941-
Byte (Byte) v. 10 (Oct. '85) p. 287-8+
DOCUMENT TYPE: Product Evaluation
ISSN: 0360-5280
LANGUAGE: English
COUNTRY OF PUBLICATION: United States
RECORD TYPE: Abstract RECORD STATUS: New record

ABSTRACT: TLC-LISP, from The LISP Company, is an outstanding microcomputer version of the popular object-oriented programming language. Using as much as 1Mbyte of memory, TLC-LISP is designed for IBM PCs and compatibles and is based on MacLISP, LISP Machine's LISP, and Logo. It supports various primitive object types, including character strings, vectors, and several types of numbers. A Smalltalk-like class system and a convenient package system make it easier to develop modules by allowing **multiple symbol tables** . TLC-LISP has a speedy, screen-based text editor. On machines with color adapter cards it supports turtle graphics. Error handling and **debugging** are well provided for, and a pseudocode compiler enables the user to create a library of compiled functions. TLC-LISP boasts fine documentation and outstanding performance. For just $250 it offers personal computer users programming features that are normally only available on sophisticated equipment.

DESCRIPTORS:
Lisp (Computer language)
Product evaluation

```
Set     Items   Description
S1       2164   (SYMBOL? ? OR SYMBOLIC()NAME? OR VARIABLE?)(2N)(TABLE? OR -
            GRID OR MATRIX OR MATRICE? OR TUPLE? OR ROW(N)COLUMN?)
S2      93565   DEBUG? OR EMULATOR? OR SIMULAT? OR MODELLING
S3         93   S1(2N)(SELECT? OR CHOOSE? OR IDENTIF? OR FIND?)
S4         49   S1(2N)(MULTIPL? OR PLURAL? OR SEVERAL? OR VARIOUS OR VARIE-
            T? OR MANY)
S5         83   S2(2N)(MULTIPROCESSOR? OR (MULTIPL? OR PLURAL? OR SEVERAL?
            OR VARIOUS OR VARIET?)(N)PROCESSOR?)
S6          6   S2 AND (S3 OR S4)
S7          5   S5 AND IC=G06F-009/44
S8          5   S7 NOT S6
S9          5   S5 AND (TABLE? OR TUPLE? OR MATRIX? OR MATRICES OR ROW(N)C-
            OLUMN?)
S10         5   S9 NOT (S6 OR S8)
File 347:JAPIO Nov 1976-2004/Apr(Updated 040802)
         (c) 2004 JPO & JAPIO
File 350:Derwent WPIX 1963-2004/UD,UM &UP=200453
         (c) 2004  Thomson Derwent
```

10/5/3 **(Item 3 from file: 347)**
DIALOG(R)File 347:JAPIO

03189740 **Image available**
DEBUGGING SYSTEM FOR **MULTIPROCESSOR**

PUB. NO.: 02-165240 [JP 2165240 A]
PUBLISHED: June 26, 1990 (19900626)
INVENTOR(s): SATO NOBUYOSHI
SAKURAI MITSUO
KOYATA SHIGENORI
IKEDA MASAHIRO
APPLICANT(s): FUJITSU LTD [000522] (A Japanese Company or Corporation), JP (Japan)
APPL. NO.: 63-319960 [JP 88319960]
FILED: December 19, 1988 (19881219)
INTL CLASS: [5] G06F-011/28; G06F-015/16
JAPIO CLASS: 45.1 (INFORMATION PROCESSING -- Arithmetic Sequence Units); 45.4 (INFORMATION PROCESSING -- Computer Applications)
JOURNAL: Section: P, Section No. 1105, Vol. 14, No. 426, Pg. 28, September 13, 1990 (19900913)

ABSTRACT

PURPOSE: To improve a debugging function by reading a stop address out of a main processor SPU, performing address conversion, and setting the address in a register of a slave processor IPU every time a TLB fault is generated.

CONSTITUTION: A stop address is stored in a register of the SPU 1 in advance and if a TLB fault occurs, the contents of the register in use is saved in the stand-by register LS of an IPU 2. Then a segment **table** and then a page **table** are read by using the current address to set an actual address in a TLB. The data saved in the LS is re-stored in the used register. Then an address stop function is checked and the stop address is read out of the SPU 1 and converted into an actual address to set in an address stop register 2a in the IPU 2. Consequently, IPUs need not inform each other of addresses and the debugging function is improved.

6/5/2 **(Item 2 from file: 347)**
DIALOG(R)File 347:JAPIO

02607027 **Image available**
MULTI-PROGRAM SYMBOL **DEBUG** SYSTEM

PUB. NO.: 63-223927 [JP 63223927 A]
PUBLISHED: September 19, 1988 (19880919)
INVENTOR(s): HASHIMOTO KAZUYA
APPLICANT(s): NEC CORP [000423] (A Japanese Company or Corporation), JP (Japan)
APPL. NO.: 62-058040 [JP 8758040]
FILED: March 13, 1987 (19870313)
INTL CLASS: [4] G06F-011/28
JAPIO CLASS: 45.1 (INFORMATION PROCESSING -- Arithmetic Sequence Units)
JAPIO KEYWORD: R131 (INFORMATION PROCESSING -- Microcomputers & Microprocessers)
JOURNAL: Section: P, Section No. 814, Vol. 13, No. 19, Pg. 124, January 18, 1989 (19890118)

ABSTRACT

PURPOSE: To efficiently execute the symbol **debugging** by designating a symbol and program discriminating information so that a symbol table can be referred to with respect to a designated program.

CONSTITUTION: At the time of starting **debugging** , by reading a symbol table into a **debugger** by a **debugger** command, a value is stored already in a program discriminating information field 104 of a symbol table 101, a symbol field 107 and a symbol attribute field 108. By a program identifier 109 designated simultaneously with a symbol 110 in a command to a **debugger** , the program identifying information fields 104, 107 and 108 in **plural symbol tables** 101-103 are retrieved, in which one coincides with the symbol 110 is selected, and the value of the symbol attribute field corresponding to its symbol field goes to symbol attribute information. In such a way, plural programs can be **debugged** by the symbol existing at every program thereof.

6/5/6 **(Item 4 from file: 350)**
DIALOG(R)File 350:Derwent WPIX

007037076
WPI Acc No: 1987-037073/198705
XRPX Acc No: N87-028068
Logic analyser using source program in trace specification - uses various symbol tables produced by compilers and provides absolute values for symbols using load map
Patent Assignee: HEWLETT-PACKARD CO (HEWP)
Inventor: GOODWIN B S
Number of Countries: 001 Number of Patents: 001
Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week
US 4636940	A	19870113	US 85814013	A	19851220	198705 B

Priority Applications (No Type Date): US 83481010 A 19830331; US 85814013 A 19851220
Patent Details:

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
US 4636940	A		39		

Abstract (Basic): US 4636940 A

The analyser has a correspondence device coupled to files associated with the linking and loading of the machine code object program, for producing a table of correspondence between symbolic labels used in the source program and corresp. absolute addresses within the relocated machine code object program that occur as states in the target system. A trace specification device coupled to the correspondence device specifies in terms of the symbolic labels a condition to be met by a logic state occurring in the target system in order for that logic state to be included in the selected trace.

A logic state acquisition device couples to the target system and acquires the values of logic states that occur and satisfy the condition specified by the trace specification device. A memory stores numerically the values of the logic states acquired by the logic state acquisition device. A converter converts from numerical values into associated symbolic labels those acquired logic state values stored in the memory which have associated symbolic labels in the table of correspondence.

USE - In **debugger** used with **emulator** .

2/6

Title Terms: LOGIC; ANALYSE; SOURCE; PROGRAM; TRACE; SPECIFICATION; VARIOUS ; SYMBOL; TABLE; PRODUCE; ABSOLUTE; VALUE; SYMBOL; LOAD; MAP
Derwent Class: T01
International Patent Class (Additional): G06F-011/34
File Segment: EPI